FreifunkFinder: An Android Application to Find the Closest Freifunk Wi-Fi Nodes

Govind Singh
Distributed Software Systems
Informatik department
TU Darmstadt
Darmstadt, Germany 64289
Email: govind.singh@
stud.tu-darmstadt.de

Puneet Arora
Distributed Software Systems
Informatik department
TU Darmstadt
Darmstadt, Germany 64289
Email: puneet.arora@stud.tu-darmstadt.de

Sooraj Mandotti
Distributed Software Systems
Informatik department
TU Darmstadt
Darmstadt, Germany 64289
Email: sooraj.mandotti@
stud.tu-darmstadt.de

***Abstract*—There are non-commercial communities available that provide free Wi-Fi facility to users, Freifunk is one such community. Such facility is preferred by users over expensive cellular networks. However, users are often not aware about availability of these free Wi-Fi nodes in his/her locality. In such situations Wi-Fi finder application comes in handy. Any such application present today generally offers Wi-fi location information on a static map which is not very user friendly. In this report, we have described a possible solution to this problem by means of an Android application. This application renders the available Freifunk Wi-Fi nodes in an augmented reality view with considerations of parameters such as distance and altitude relative to the current location of the user. The Application also has user configurable parameters that limits the scan radius and number of nodes displayed on the screen. The application in its current form is customized to use Freifunk Darmstadt Wi-Fi nodes, however it can be extended to be used anywhere with minor modifications due to its flexibility.**

***Index Terms*—Freifunk, Wi-Fi nodes**

I. Introduction

Freifunk is a non-commercial initiative for free wireless networks. Any user can access free Wi-Fi provided by Freifunk community volunteers. In return, user in the free wireless network can provides his or her wireless LAN router for data transfer to other participants. Any user can also transmit data, such as text, music through a free internal network or use services set-up by participants to chat, call or play on-line games. Many users in freifunk community share their internet access and allow others to use it to access the World Wide Web. Community users use freifunk firmware a special Linux distribution, on their WLAN routers. There are many Freifunk communities that meet regularly to co-ordinate this project in order to suppy Freifunk Wi-Fi coverage in villages and cities. The freifunk community is part of a global movement for free infrastructure and open frequencies. Their vision is the democratization of the media through free networks. Free wireless communities implement this idea worldwide [1]. In this application(called FreifunkFinder), we developed an augmented reality Wi-Fi nodes finder that locates the available Wi-Fi nodes based on the Wi-Fi nodes information provided by Darmstadt Freifunk community.

Augmented reality (AR) is a live, direct or indirect, view of a physical, real-world environment whose elements are augmented by computer-generated sensory input such as sound, video, graphics or GPS data. It is related to a more general concept called mediated reality, in which a view of reality is modified (possibly even diminished rather than augmented) by a computer. As a result, the technology functions by enhancing ones current perception of reality. By contrast, virtual reality replaces the real world with a simulated one. Augmentation is conventionally in real-time and in semantic context with environmental elements, such as sports scores on TV during a match. With the help of advanced AR technology (e.g. adding computer vision and object recognition) the information about the surrounding real world of the user becomes interactive and digitally manipulable. Artificial information about the environment and its objects can be overlaid on the real world [2].

One such example of Augmented Reality is AR Browser where user can select multiple layers of content to place into his/her AR browser allowing user to automatically find detailed content based on what user is pointing his/her phone at. If user finds what he/she needs, he/she can click to get more information on almost anything [3].

For our application, available Wi-Fi nodes information is collected using Freifunk Darmstadt server. Our application makes an HTTP call to the Freifunk server at the start of the application or after user pushes the refresh button (possible future extension). This call prompts a reply from the Freifunk server with file in JSON format containing information on the Freifunk nodes registered with the requested community. The received node information contains a number of parameters, from which a selected subset is displayed in the application. Gathered information is parsed, filtered and stored in application's database from where it is read multiple times during the application run later on.

After storing the available node information, application makes use of user's location information (provided by GPS or Wi-Fi/mobile networks, but preferably GPS) and also sensor information provided by Android APIs. Our application requires access to the following device features:

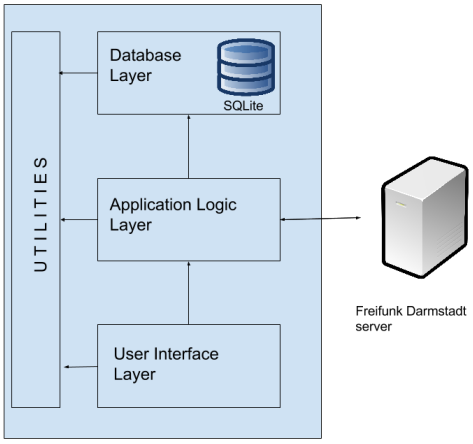


Fig. 1: A design for the application.

- Camera.
- Location provider.
- Compass and Accelerometer sensors.

Later, these sensor information is used to display Wi-Fi nodes in an augmented reality mode, which means the user just points his/her camera in a direction and Freifunk nodes (if available in that camera view) are displayed on top of the image recorded by camera.

In the market a few Wi-Fi finder applications are available on the Android platform, however location and altitude parameters are often ignored by these applications [4] [5]. Also, the majority of these applications render available Wi-Fi access points only on a static map which makes it difficult for the user to identify the exact location of a specific access point. These shortcomings of existing applications drove us to create a free Wi-Fi finder application for Freifunk network that takes into account of the distance parameter (including altitude) and displays it in an augmented reality fashion to make it a user friendly application of its kind.

II. Design

The Fig. 1 shows the high-level design of the application. Our application follows layered architecture consisting of an User Interface layer, Application Logic layer, Database layer and a few Common Utilities used by all the layers. Application logic layer and User Interface layer are core of the application. Application layer acts as a glue between User Interface and Database layer and performs all domain-related functionalities. These functionalities include reading and interpreting data from Freifunk server, distance calculation between users current location and available Wi-Fi nodes, sorting of nodes, filtering of nodes based on user configuration etc.. This layer also provides interfaces for external entities to interact with it. The distance calculation is done using Haversine distance calculation formula [6] as it gives more accurate results for the distance between two points on Earth's surface. The sorting of Wi-Fi nodes is based on their distances from the user's current location and a sorted list is then forwarded to the User Interface layer

The User Interface layer is crucial in terms of user interaction and handling of user generated events during the application run. It performs the complicated task of rendering the sorted Wi-Fi nodes (sent by above layers) on the device screen in an efficient manner so that the glitches observed are minimal. Other tasks handled by this layer include user configuration handling, reading and processing of sensor-related parameters, user location management, screen shuffling, Wi-Fi node icons rendering etc.. Handling of sensor-generated events in real-time gets cumbersome due to its inherent nature of very high frequency in which these events are fired and expected to be handled. User Interface layer successfully overcome it by using (tweaked) sensor smoothing algorithms. Also, using both GPS and Network provider and swapping efficiently between these two as per the availability, this layer manages to extract the best location related information in real time. The priority is always given to GPS provider for obvious reason of height information associated with it. The rendering of Wi-Fi node icons depending upon node's closeness to the user location, augments the reality with node's precise location information and present it in a user-friendly manner.

Through the DB layer Wi-Fi access point information is persisted in Android's SQLite database to be used in later scenarios. Existing Android APIs are used to perform the typical database operations. If application is re-started within short span of time then Freifunk available nodes information is read from the local database itself else information is requested from Freifunk server. This prevents the inherent delay that's inevitable when reading data from external entities. The utility layer is light-weight and contains the model object that is transferred across the layers, along with some objects which are meant to be globally available throughout the application [7] [8].

III. User Interface

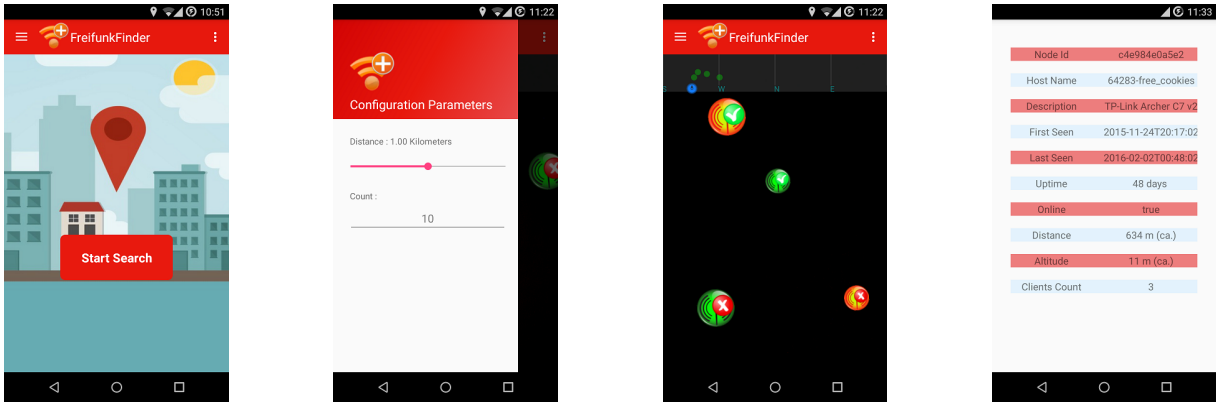
The application consists of three screens namely User Configuration screen or Home screen, Rendering screen, Node Detail screen.

A. Home screen

This screen is shown when user starts the application(see Fig. 2). It consists of flow (vertical) layout with elements such as application logo and a button that starts search for available Wi-Fi nodes.

As shown in Fig. 3, on click of options menu following user configurable parameters are displayed.

- Distance- It represents the scan distance of the application i.e. if device is pointed in a particular direction then upto what 'distance' available Wi-Fi access points will be displayed. Maximum value for distance parameter is 2 Km and default value is 1 Km.
- Count- It represents the number of available Wi-Fi nodes that are displayed on screen. If there are more Wi-Fi nodes than this parameter value then sorting is done based



(a) Home screen (b) User configurations (c) Available Wi-Fi nodes (d) Specific node information

Fig. 2: User Interface

on distance and top results are displayed. Default value is set to 10.

B. Rendering screen

User is navigated to this screen when user selects 'Start Search' on Home screen. This screen has two separate views, Overlay view and Camera view. In the latter view, device's camera view is used with the help of Android Camera API. For this application requires access to user's camera. In the Overlay view available Wi-Fi nodes from Freifunk are rendered onto the location the camera is currently pointed at. Available node information is fetched from Freifunk server while user is at home screen and persisted in the application database file. At the time of rendering onto screen required nodes information is fetched from database only.
Fig. 4 represents the Overlay view with available Wi-Fi nodes. As shown, this screen also has a one dimensional translucent compass placed on top of the screen with a small blue indicator pointing to the device direction and also mini nodes indicator that tells which direction has nodes.

In Fig. 4 you can see Overlay view depicting available Wi-Fi node icons in this camera focus on the overlay.The overlay is dynamically updated once the user moves its camera focus in another direction.

C. Node detail screen

User is navigated to this screen when user selects any particular node out of all nodes displayed on the overlay view on screen. This screen renders all the relevant information about the node selected. All of this information is retrieved from Freifunk Darmstadt server in a file of JSON format which is parsed and displayed in user friendly manner. Following information is displayed on screen:

1) Node Id: Selected Wi-Fi node Id provided by Freifunk.
2) Host name : Selected Wi-Fi node name as registered with Freifunk.
3) Host description: Descriptive information about the node selected as retrieved from Freifunk.
4) First Seen: It shows when the node was first registered with Freifunk.
5) Last Seen: It shows when was this node seen for the last time.
6) Uptime: Time since this node is active.
7) On-line: Whether the node is on-line or off-line.
8) Distance from user: As calculated by the application.
9) Altitude: Height information of the specific node.
10) Clients Count: Number of clients currently connected with this node.

IV. Implementation

The core logic of this application is to calculate current camera view of the user's device and display Wi-Fi nodes if they lie within the camera view based on the latitude, longitude and altitude information of the Wi-Fi nodes.

A. Wi-Fi nodes information retrieval and persistence

Retrieval: Our application depends on nodes information which are later displayed in an augmented reality view. So retrieving the node information from an external server is most important step in the application work cycle. Nodes information is retrieved by making a HTTP request call to Freifunk Darmstadt server. This call is replied with a JSON object. Our application handles this JSON object and parses it to extract information out of it. Nodes information is then passed on to lower layer.

Persistence: Imagine a scenario where user restarts the application multiple times without change in its location and within a short span of time. If nodes information retrieved in first start of the application is not persisted then application is going to make a HTTP call to an external server with each start of the application which is expensive in terms of both data and performance. We have avoided such frequent HTTP calls by using an already available light-weight database in Android devices called SQLite. So application makes HTTP call to Freifunk Darmstadt server only if the user's location has changed considerably (future extension) or application is restarted after a short span of time.

B. Application Managers

1) Android Sensors overview: Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing [9].

The Android platform supports three categories of sensors:

1) Motion sensors- These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
2) Environmental sensors-These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
3) Position sensors-These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers [9].

Our application makes use of Motion sensors and Position sensor namely accelerometer and magnetometer.

2) Mobile Sensor Manager: Android provides SensorEventListener interface which can be implemented so as to listen for events like accuracy change and sensor change by merely overriding methods onAccuracyChanged() and onSensorChanged(). However we are only interested in onSensorChanged(), which is called whenever there is sensor change event happens and give us new sensor values. We refresh and redraw the OverlayView for each sensor change event [10].

3) Mobile Location Manager: Role of Mobile location manager is to keep track device's geographic position. Android provides an interface called LocationListener that provides method declaration to listen for device's location based on Network or GPS provider. For this purpose developer has to override onLocationChanged() method which require a location provider. For reliability purpose we implemented this interface twice, one implementation uses GPS as provider for onLocationChanged() method while other uses Network. If both are available as provider, application gives preference to GPS over Wi-Fi/Network. Method onLocationChanged() gives new location based on time or distance and update MobileLocation global variable. For our application we chose time as the criteria for update of location information with event triggering frequency of 50ms [10].

We unregistered sensors and remove location update when the application is on pause. This saves the battery.

C. Sensor reading smoothing

One of the challenge in using sensor recorded values is that the values are too frequent with lots of noise. Noise refers to disturbance recorded by the sensor making its value to fluctuate very frequently. So we had to smooth-en sensor readings. smoothing was a two step process:

1) Averaging [11]
2) Exponential smoothing [12]

Selection of exponential coefficient depends on average value of the sensor readings over last 5 recordings. Sensors record and report new sensor reading very frequently (say every 100 millisecond) so to replace old value with new ones we used modulo 5 function as we are averaging over last 5 values. If the average value is less than a threshold value (1.5 in this application) we believe camera position is relatively stable hence a low exponential coefficient is selected(0.03) and if the average value of last 5 sensor reading lies above the threshold value a high exponential coefficient is chosen (0.8). Consider following cases:

1) Case 1: When the device moves swiftly (sensor fluctuations are high) In such a scenario, more weightage is given to the new sensor reading. Hence, a high coefficient value (0.8) is preferable.
2) Case 2: When the device movement is slow (sensor fluctuations are low) In such a scenario, since sensor values are not drastically changed, thus keeping a low coefficient value (0.03) would suffice.

After selecting exponential coefficient we used a simple formula to smooth-en the new value:

$$\text{newValue} = \text{oldValue} + \text{ex.coefficient} * (\text{newValue} - \text{oldValue})$$

As a result of above formula, the obtained smooth-en value always evaluated in accordance with the amount of fluctuations and hence keeping it relatively stable.

D. Encapsulating camera view on a custom view

We have used a FrameLayout for Wi-Fi node rendering screen. A FrameLayout is most appropriate, as it allows the layering of views within the display area. We defined a custom view control for use within this FrameLayout. Androids SurfaceView class can be used for this purpose. Therefore, we used this class to encapsulate our camera drawing. Further, implementing Android's SurfaceHolder callbacks provides a way to customise surface's life cycle by overriding methods like surfaceCreated(), surfaceChanged(), and surfaceDestroyed(). In surfaceCreated() callback method we request the camera, set the camera's display orientation based on the current orientation of the device (so that the camera preview always displays right-side up), and tie the camera preview to the SurfaceHolder by calling Android's setPreviewDisplay() method. In the surfaceChanged() callback we request the camera parameters and check what the supported preview sizes are. It is necessary to find a preview size that can be

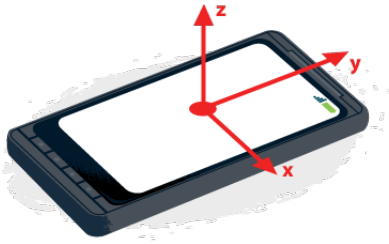


Fig. 3: Device co-ordinate system

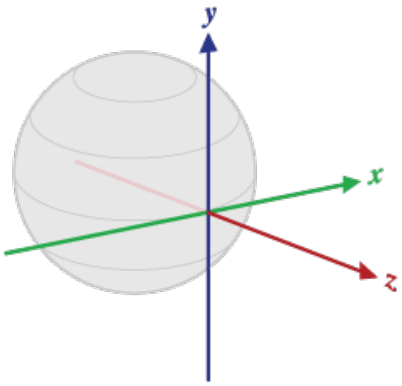


Fig. 4: World co-ordinate system

accommodated by the surface, so we selected the closest match and used it as our preview size. Next, we set the camera preview format, commit the camera parameter changes, and finally called the startPreview() method to begin displaying the camera preview within the surface. We wrapped up with the surfaceDestroyed() callback method. Here we shut down the camera preview and release the Camera resources [10].

E. Node placement calculation on Overlay view

Another important part of implementation was Overlaying Wi-Fi node icons on top of the camera preview. We have done this by implementing a custom view that can be layered on top of the camera view currently displayed in the FrameLayout control of the activity. So we extended Android's View class which implements onDraw() method. This onDraw method is called after each invalidate call and is where we did our real work, drawing Wi-Fi nodes. Before determining positions of each relevant Wi-Fi nodes we have to determine the device's orientation. Android's SensorManager class provides efficient ways to calculate this information through methods like getRotationMatrix(), remapCoordinateSystem() and getOrientation(). A brief explanation of these methods are as follows [10].

getRotationMatrix() : Computes the inclination matrix I as well as the rotation matrix R transforming a vector from the device coordinate system(Fig. 6) to the world's coordinate system (Fig. 7) which is defined as a direct orthonormal basis. This method uses device's Accelerometer and Magnetometer sensor readings [9] [13].

remapCoordinateSystem() : Rotates the supplied rotation matrix so it is expressed in a different coordinate system. Our application uses this method to remap Y axis along the camera's axis.

getOrientation() : Computes the device's orientation based on the rotation matrix.

Based on the above obtained device orientation we draw relevant Wi-Fi nodes in a canvas. For the purpose of our application we keep this canvas in a fixed position. We move this canvas (using Android's translate() method) whenever user rotate the device by dx and dy amounts around Y axis and X axis respectively.

There are three kinds of rotation for a device to be noted:

1) Azimuth : rotation around the -Z axis, i.e. the opposite direction of Z axis.
2) Pitch : rotation around the -X axis, i.e the opposite direction of X axis.
3) Roll : rotation around the Y axis.

Calculating Wi-Fi nodes position:

1) Calculating X position : We find bearing to each of these relevant Wi-Fi nodes based on its location and device's location. Then we calculate the node's X position form actual canvas position.

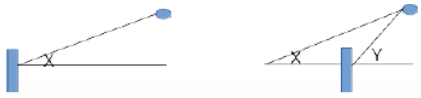


Fig. 5: Distance calculation

2) Calculating Y position : Here we need to explicitly find the vertical angle of each node from horizontal line going through the device's position So if we move closer to a node the angle increases (Y is greater than X). As a result we also increase the position of nodes dynamically in our drawing as we move closer.

F. Haversine distance calculation

The great-circle or orthodromic distance is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere (as opposed to a straight line through the sphere's interior).Through any two points on a sphere which are not directly opposite each other, there is a unique great circle. The two points separate the great circle into two arcs. The length of the shorter arc is the great-circle distance between the points [14].

Let ϕ_1, λ_1 and ϕ_2 , λ_2 be the geographical latitude and longitude of two points 1 and 2, and $\Delta\phi$, $\Delta\lambda$ their absolute differences; then Δ σ, the central angle between them, is

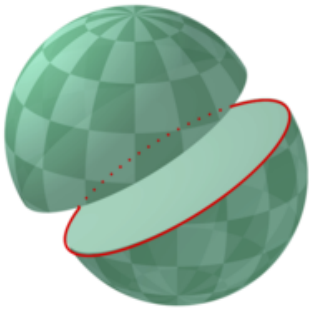


Fig. 6: Sphere depicting great circle

given by following Haversine formula [6] :

$$\Delta\sigma = 2arctan\sqrt{sin^2(\frac{\Delta\phi}{2}) + \cos\phi_1 . \cos\phi_2 . \sin^2(\frac{\Delta\lambda}{2})}$$

The distance d, i.e. the arc length, for a sphere of radius r and $\Delta\sigma$ given in radians

$d = r\Delta\sigma$

G. Rendering Wi-Fi nodes on overlay view

To enhance the user experience displayed Wi-Fi nodes are color coded and dynamic in size and comes along with a symbol to show altitude information correctness. If the node lies within half the value of distance selected by user the node icon is big and if it lies in the other half we used a smaller display icon for it. For few of the nodes altitude information is not provided by Freifunk Darmstadt server. To distinguish such nodes from the ones that have correct altitude information we used a tick and a cross icon on the node. Stating the obvious, tick represent the nodes with correct altitude information and cross suggests no altitude information of this node with the application. Finally on-line and off-line nodes are displayed in different colors. "On touching a Wi-Fi node icon the touch position is map onto a position in canvas of the Overlay view. Based on this mapping we find the most nearest Wi-Fi node from touch position from the list of current displaying Wi-Fi nodes [15].

H. Call to Database Layer

After above distance calculation call to below layer is made under following conditions (for getting relevant Wi-Fi nodes) if:

1) user changed any of the configurable parameters.
2) user position changed more than permissible limit from last saved position (limit currently set to 2 meters).
3) Refresh button is selected(This button can be used to refresh HTTP call to Freifunk Darmstadt. It is currently a future extension scope item)

V. External Entities

A. Freifunk Darmstadt Server

Our application relies on the Freifunk community server to get the information about the available Wi-Fi nodes. So at the start of the application HTTP request call is made to Freifunk Darmstadt server to retrieve the Wi-Fi node information. The server replies with a file with JSON format which is parsed by our application and node information is retrieved. However to improve efficiency of the application we allow HTTP call only in a scenario when:

1) Application is started for the first time.

OR

1) User position has changed more than threshold value.

OR

2) Application is restarted after some time.

Above steps reduces HTTP call traffic to Freifunk Darmstadt server and increase application performance as data is read from local SQLite DB instead.

VI. Limitations

There are few limitations of the application that are worth mentioning:

1) Application is designed to work only in portrait mode i.e only when user holds the phone in upright position. So if the user tries to find Wi-Fi nodes in landscape mode, then node distance and altitude displayed on screen will not be accurate.
2) Application can select best location provider (GPS, Network or Wi-Fi) automatically on its launch however, if the application is using network as location provider then it will not give altitude information so, as a workaround altitude is set as 150.0 meters as default. Hence Node icons displaying altitude information in these cases will not be correct.

VII. Future extension

1) Dynamic Wi-Fi strength rendering: Currently while displaying the Wi-Fi node onto the screen Wi-Fi signal strength parameter is not taken into account. However, we propose a future extension which renders Wi-Fi signal strength information dynamically. So, if the user is moving away from a particular Wi-Fi node, the corresponding Wi-Fi node icon changes dynamically to display weaker signal strength.

A. Sorting based on Wi-Fi strength

As our application provides flexibility for the user to limit the number of available Wi-Fi nodes in camera view of the user, hence filtering is needed in selecting the most appropriate nodes out of all available Wi-Fi nodes. Currently this selection is done on the basis of distance between the user and node. We propose having Wi-Fi strength as one of the possible selection criteria. So if the Wi-Fi node that is far from user having better Wi-Fi strength will be preferred over the node that is closer to the user(but with weak Wi-Fi signal strength).

B. Location based information display

Currently our application is relying on Freifunk Darmstadt server to retrieve information about the available Wi-Fi nodes. In future our application can be modified easily to display information about available nodes outside of Darmstadt. This would be possible by mere change of URL from Freifunk Darmstadt server to required location Freifunk server.

C. Information refresh button

This button can be used to refresh the information currently saved in the DB. On selection of this button a fresh HTTP request is sent to Freifunk Darmstadt server and updated nodes information can be fetched.

VIII. Conclusion

As part of this task, we successfully developed an android application that displays the available Wi-Fi nodes in the neighbourhood in an augmented reality fashion. while doing so this application uses all three of the location parameter(latitude,longitude, altitude) to render available nodes on the overlay view while user's camera view is in background. This application also provide user configurations to modify application search parameters as per his/her convenience. The design of the application is flexible enough for future extensions.

Acknowledgements

The authors would like to thank ETIT department for providing the opportunity. We would also like to personally thank Mr. Fabian Kaup for his help and guidance throughout our work. We would like to show our gratitude towards Freifunk Darmstadt community for promptly replying to our queries regarding Freifunk services. The background image for the application is designed by Freepik.com

References

[1] "Freifunk darmstadt," https://wiki.freifunk.net/, [Online; Accessed: 17-January-2016].

[2] "Augmented reality," http://www.mashable.com/category/augmented-reality, 2014, [Online; Accessed: 27-January-2016].

[3] A. Hepburn, "Top 10 augmented reality examples," http://www.digitalbuzzblog.com/top-10-augmented-reality-examples, 2010, [Online; Accessed: 1-February-2016].

[4] "Wifi finder," https://play.google.com/store/apps/details?id=com.jiwire.android.finder\&hl=en, [Online; Accessed: 21-November-2015].

[5] "Wifimaps: free wifi +passwords," https://play.google.com/store/apps/details?id=com.dzencake.wifimap, [Online; Accessed: 21-November-2015].

[6] "Haversine formula," http://www.mathforum.org/library/drmath/view/51879.html, 20-April-1999, [Online; Accessed: 30-January-2016].

[7] "Android-sqlite-query," http://www.mysamplecode.com/2011/10/android-sqlite-query-example-selection.html, [Online; Accessed: 21-December-2015].

[8] "Wifi finder," http://www.vogella.com/tutorials/AndroidSQLite/article.html, [Online; Accessed: 11-November-2015].

[9] "Android sensors," http://developer.android.com/reference/android/hardware/SensorManager.html, 2014, [Online; Accessed: 27-November-2015].

[10] L. Darcey and S. Conder, "Android sdk augmented reality: Location and distance," http://code.tutsplus.com/tutorials/android-sdk-augmented-reality-location-distance--mobile-8004, 30-September-2011, [Online; Accessed: 1-November-2015].

[11] "Sensor smoothing - averaging," http://stackoverflow.com/questions/15864223/, 20-April-1999, [Online; Accessed: 30-January-2016].

[12] "Sensor smoothing using exponential coefficient," http://stackoverflow.com/questions/7598574, 20-April-1999, [Online; Accessed: 30-January-2016].

[13] "Image credits," http://www.leesicons.com/north-compass-icons.html,https://openclipart.org/detail/10941/red-+-green-ok-not-ok-icons,http://www.clipartbest.com/clipart-xcgK5qdai,http://icones.pro/ajouter-en-plus-radio-wifi-image-png.html, [Online; Accessed: 21-November-2015].

[14] "Great circle," https://en.wikipedia.org/wiki/Great-circle_distance, 15-April-2015, [Online; Accessed: 20-January-2016].

[15] "smoothing-data-from-a-sensor," http://stackoverflow.com/questions/4611599/help-smoothing-data-from-a-sensor, [Online; Accessed: 21-November-2015].